International Journal of Computer Applications (0975 – 8887)
Volume 148 – No.14, August 2016

A Comparative Analysis of Clone Detection Tools: Solid SDD and CCFinderX

Muhammad Ilyas
Department of
Computer Science and IT,
University of Sargodha,
Sargodha, Pakistan

Hafiz Anas Bilal
COMSATS Institute of
Information Technology,
Islamabad, Pakistan

Muhammad Hummayun
Department of
Computer Science and IT,
University of Sargodha,
Sargodha, Pakistan.

Mubeen Rafi
COMSATS Institute of
Information Technology
Lahore, Pakistan

Almas Kanwal
Department of
Computer Science and IT,
University of Sargodha,
Pakistan

ABSTRACT
There are several tools available for code clones detection and removal. Over the last few years much research has been done on assessment of these tools. Every tool has its efficiencies and deficiencies which researchers tried to evaluate. But the imperative point that we observed while analyzing these assessments is that there is no benchmark defined in this context so far. There is no clear picture that depicts which tool is better than the other and why? This paper is a contribution in this scaffold. Two clone detection tools SolidSDD and CCFinderX are evaluated and a comparison of these two is on hand here. Some experiments are performed on an open source software i.e. VLC media player, and it is revealed how different clone detection tools provide different results when study the same system. Reasons for these variations in results are endeavor to find out at this juncture.

General Terms
Code cloning, clone detection

Keywords
Code cloning, clone detection

1. INTRODUCTION
Code cloning is an aged practice in programming framework that attains short term advantages of timely completion of projects and dodging of code rewriting. These short term advantages have to pay later when a system needs to evolve. Code clones also make the maintenance task tidy but not every clone need to be removed from the code instead the clones whose removal achieves maximum gain (in terms of resource utilization) can be targeted.

Enough literature is available on code clone perspective and need for removing these clones from the code but no standardized tools are available nor any specific parameters are defined so that architect target only those parameters for clone removal. In this paper a comparison between two clone detection tools is spotlighted. It is shown how both tools vary in detection methodology and show variations in results. Though these variations do not cause any special effects if only used for analysis purpose but arise many questions when intention is removing the clones after detection. It becomes difficult to decide which tool's result should one considered.

To be evidence for the above statement two clone detection tools SolidSDD and CCFinderX are used and an experimental study is conducted on an open source project. Comparison is not only performed on whole project rather a single file is chosen randomly and results are analyzed in greater detail.

Rest of the paper is structured as follows: at first there is relevant literature, and then a case study is presented. After that a comparative study is performed. At the last there is conclusion of the analysis and some facets of future work.

2. LITERATURE REVIEW
The clone presence in the code makes the software system more complex and adds to maintenance cost [2]. A clone pedigree was built in [3]. Up till now three types of clones are identified in the literature i.e. type1 are exact identical clones, type 2 clones are those that become identical after variable renaming and type 3 clones turn into clones but after adding, deleting and modifying some lines of code.

Human intuition is good judgment for sensing clones but of course with this approach scalability issue arises. So, automatic clone detection and removal tools are developed. A tool that provides clone detection feature may not endows with removal functionality. Some of the refactoring techniques also used for clone elimination [1].

A comparison is presented in [4]. Two tools CONQAT and SolidSDD are used for clone detection. Characteristics of both tools are separated in the paper and the comparison was performed on six parameters that exist in both tools. It was concluded that SolidSDD is better than the other in terms of taking less time and finding out more clones. Though this paper is precise but there is no information at what scale the code is experimented and where the differences lie in both tools.

A similar writing is available in [5]. Three clone detections techniques i.e. simple line matching parameterized matching and metric fingerprinting were compared. Focus of the paper was to find out which technique is more suitable for specific task. It was concluded that line matching only gives indication of clones. Parameterized matching is well suited if used with refactoring tools that work on statement level. Method level refactoring tools work best with metric fingerprinting.

3. CASE STUDY

This section evaluates the both tools used in this research work. The setting up of experiments also discussed here.

3.1 Tools Evaluation

The under discussion tools i.e. SolidSDD and CCFinderX are assessed in this section.

3.2 SolidSDD

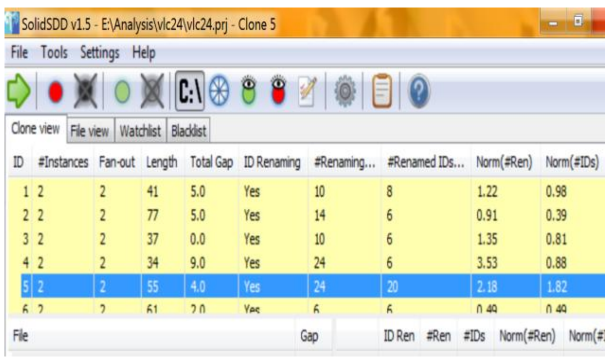
The first tool that is chosen for this case study is SolidSDD (version 1.5). It provides multi lingual support. Else then its lenience to variation, scalability, speed, ease of integration and configuration the most attractive feature of this tool is its simplicity of use (user friendliness). Another very interesting feature is that once the target project has been loaded, its integrity is verified every time it is reloaded i.e. if some statements or clone has been moved or changed SolidSDD tracks that changes and notify to user. An eye-catching interface is provided. Visual synopsis of detection results is also provided that put in the picture the distribution of clones in the software. Reports of the detection result can also be easily generated.

To find out the clones at first the user is required to have a name for new analysis project, specify the location of Source folder. Specify the Output folder. and adjust the analysis setting i.e. set values of Local Gap (statements added, deleted or modified while copying and pasting code), Cumulative Gap (sum of all local gaps), Gap Decay (decrease in local gap), and Minimum clone size parameters.

The detection process is comprised of two main steps that are pre-processing and extraction. In pre-processing step features that will be used for detected clones are alienated. Extraction step is further subdivided into four sub steps that are: Initializing clone detector, Finding clones, cleaning up and Post processing clones.

After completing these two steps a brief summary is displayed containing total number of clones found and the execution time taken by this process. Detailed view has four tabs that are Clone view, File view, Watchlist and Blacklist.

Each of these four tabs is briefly discussed here. First of all Clone View of SolidSDD is illustrated in Figure 1.


SolidSDD v1.5 - E:\Analysis\vlc24\vlc24.prj - Clone 5

ID	#Instances	Fan-out	Length	Total Gap	ID Renaming	#Renaming...	#Renamed IDs...	Norm(#Ren)	Norm(#IDs)
1	2	2	41	5.0	Yes	10	8	1.22	0.98
2	2	2	77	5.0	Yes	14	6	0.91	0.39
3	2	2	37	0.0	Yes	10	6	1.35	0.81
4	2	2	34	9.0	Yes	24	6	3.53	0.88
5	2	2	55	4.0	Yes	24	20	2.18	1.82
6	2	2	61	2.0	Yes	6	6	0.49	0.49

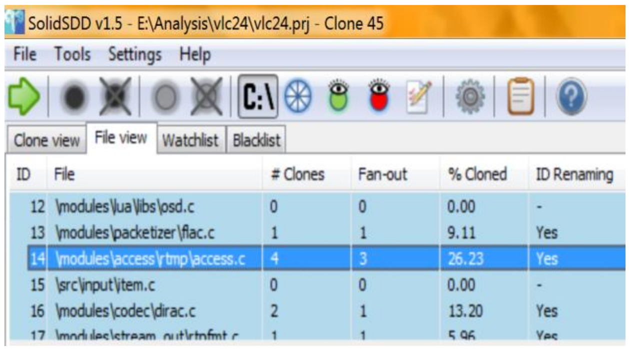
File	Gap		ID Ren	#Ren	#IDs	Norm(#Ren)	Norm(#

Fig 1: Solid SDD clone view

This view has several terms that are briefly described below:

- Clone ID: An integer assigned to each clone occurrence.
- No. of Instances: Number of that particular clone instances.
- Fan out: Count of files containing instances of a particular clone.
- Length: Length of clone instance.
- Total Gap: Total number of statements placed in gaps of instances of clone code.
- ID Renaming: This parameter gives information that whether identifiers, variables have been renamed, partially renamed or not renamed.
- #Renamings (#Ren): The average number of renaming that has been detected for each cloning relation that displays renaming.
- #Renamed IDs (#IDs): Average number of unique identifiers that have been renamed
- Norm (#Ren): The average number of renaming normalized by the length of the corresponding clone instance
- Norm (#IDs): Average number of unique identifiers that have been renamed normalized by the length of the corresponding clone instance.

Second tab is File view as shown in Figure 2. The parameters defined in this view are as follows:


SolidSDD v1.5 - E:\Analysis\vlc24\vlc24.prj - Clone 45

ID	File	# Clones	Fan-out	% Cloned	ID Renaming
12	\modules\lua\libs\osd.c	0	0	0.00	-
13	\modules\packetizer\flac.c	1	1	9.11	Yes
14	\modules\access\rtmp\access.c	4	3	26.23	Yes
15	\src\input\item.c	0	0	0.00	-
16	\modules\codec\dirac.c	2	1	13.20	Yes
17	\modules\stream_out\rtpfmt.c	1	1	5.96	Yes

Fig 2: Solid SDD file view

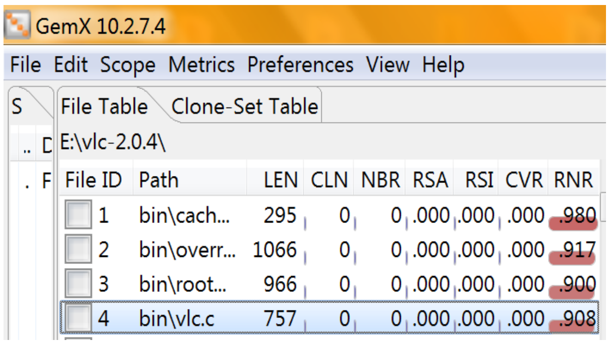
- #Clones: Total number of detected clones in particular file.
- Fan out: Number of files with which cloned fragment has cloning relation.
- % cloned: Percentage of statements in clone instance.
- ID Renaming: It tells about whether identifiers, variables have been renamed, partially renamed or not renamed. The last two tabs are Watchlist and Blacklist. Watch list contains the clones in which we are more interested while black list includes non-interested clones.

3.3 CCFinderX

CCFinderX is another tool chosen for this research. It detects clones using two steps: In first step user has to select the language and the targeted software. In second step user specify certain parameters i.e. Minimum Clone Length, Minimum Token Size (Minimum TKS), Shaper, P-match (parameterized matching) and prescreening.

GemX interface displays the result using two windowpanes i.e. right and left. Right pane contains scatter plot, source text and scrapbook tabs. Scatter plot is a graphical representation that tells about scatteress of clones in the source code. Source text tab shows actual source code of the underlying software. Cloned code is highlighted within the file. Scrapbook is just like a clipboard on which code can be kept temporarily.

Left window pane has two important tables namely file table and clone set table. Tab containing file table is shown in below Figure 3.and termed are defined as:

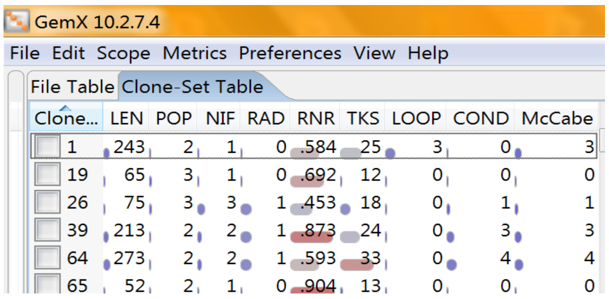


File ID	Path	LEN	CLN	NBR	RSA	RSI	CVR	RNR
1	bin\cach...	295	0	0	.000	.000	.000	.980
2	bin\overr...	1066	0	0	.000	.000	.000	.917
3	bin\root...	966	0	0	.000	.000	.000	.900
4	bin\vlc.c	757	0	0	.000	.000	.000	.908

Fig 3: CCFinderX File View

- File ID: An integer value assigned to clone instance.
- Path: location of the selected file with complete hierarchy.
- LEN: size (in tokens) of selected file.
- CLN: It is the count of clones in the files.
- NBR: Count of files having a particular clone code portion
- RSA: It is defined as Percentage of tokens that are covered by a code clone between the file and one of the other files.
- RSI: Ratio of Similarity with in the file is percentage of tokens that are covered by a code clone within the file.
- CVR: Coverage of clone is defined as Percentage of tokens that are covered by any code clone
- RNR: Ration of Non Repeated tokens is defined as 1-(Ratio of Repeated tokens)

The other tab in left tab is Clone-Set Table that is shown in figure 4.



Clone...	LEN	POP	NIF	RAD	RNR	TKS	LOOP	COND	McCabe
1	243	2	1	0	.584	25	3	0	3
19	65	3	1	0	.692	12	0	0	0
26	75	3	3	1	.453	18	0	1	1
39	213	2	2	1	.873	24	0	3	3
64	273	2	2	1	.593	33	0	4	4
65	52	2	1	0	.904	13	0	0	0

Fig 4: CCFinderX Clone View

This table provides the following information:

- Clone-Set ID: Integer value assigned to each clone instance.
- LEN: Size (in tokens) of cloned code fragment.
- POP: It is number of instances of particular clone instance.
- NIF: Count of source files that include one or more code fragments of the code clone.
- RAD: Range of source code fragments of a code clone
- RNR: Ratio of non repeated tokens is the percentage of tokens that are not included in repeated part of code fragment
- TKS: Token set size of a code fragment of the code clone.
- LOOP: Count of loops in a code fragment.
- COND: Count of conditional branches
- McCabe: McCabe is sum of LOOP and COND.

3.4 Experimental Setup
The system (machine) on which study is conducted and the software system (subject system) that is used for analysis are introduced here.

3.4.1 The Subject System
VLC Media player is Simple, fast and powerful media player. It is chosen as a subject system. VLC is developed in C. It plays multimedia files and everything like, Discs, Webcams, Devices and Streams, and most codec's with no codec packs needed. Besides playing many files it can do media conversion and streaming. It's an open source project having 6227707 LOC and 321941 SLOC (source lines of code). Several versions of VLC media player are available. Version 2.0.4 is used here for experimentation.

4. COMPARATIVE STUDY
This section illustrates the tools characteristics and results of various experiments.

4.1 Experimental Setup
Before comparing the functional methodology and result variations, basic characteristics of both tools must be considered. These can be supportive in general assessment of both tools. These characteristics are shown in table1.

It is very clear from the results that both tools have competitive features. So, it would be beneficial to note down the variations in results that both tools revealed while experimented on same system. This task is accomplished in subsequent breakdown.

4.2 Experimental Results
Experiment is conducted on VLC media player and outcomes are observed on many different parameters. Results are further analyzed on individual files because SolidSDD did not give detailed information on whole project however it gives detailed information when individual files are evaluated. Furthermore, as both tools uses different clone detection methodologies i.e. one uses token based and other uses string based approach so default values are used for input parameters in both detection software. First experiment is carried out using SolidSDD with following input parameters:

- Minimum Clone Size (CS) = 35,
- Cumulative Gap (CG) = 2.5,
- Gap Decay (GD) = 0.1
- Local Gap (LG) = 2

Results are shown in table 2.

SolidSDD is further tested by changing values of parameters.

CS= 35; GD= 0.1; CG= 2.5 are set aside constants while LG varies through 0 to 50.

Result of these variations is shown in below graphical form.

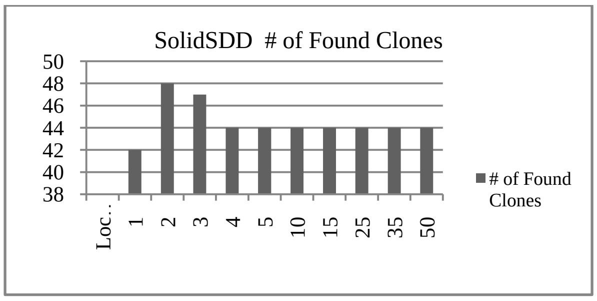


Fig 5: No. of found clones with SolidSDD on varying Local Gap

It can be seen from the graph that results vary only for first three value of local gap. Remaining output is constant for rest of the values. It gives high clone ratio at local gap value 2 and 3.

After experimentation with SolidSDD, the same scenario is repeated with CCFinderX. Input parameters are assigned values as follows:

- Minimum Clone Length (CL) = 50
- Minimum Token Set Size (MTK) = 12
- P-Matching (PM) = Yes
- Shaper (SH) = Soft

Output is shown in table 3.

Results are further generalized by varying parameter values. Minimum Clone Length(CL), Minimum Token Set Size (MTK), P-Matching (PM) are set as constant and Shaper(SH) is set as variable that can take one of four values i.e. Easy, Hard, Soft and No Shaper. Each of these has been assigned integer values that are:

- Easy = 0; Hard = 1; Soft = 2; No = 3;

  Constants are assigned values as

- CL = 35; MTK = 12; PM = Yes

The output is shown in below graph:

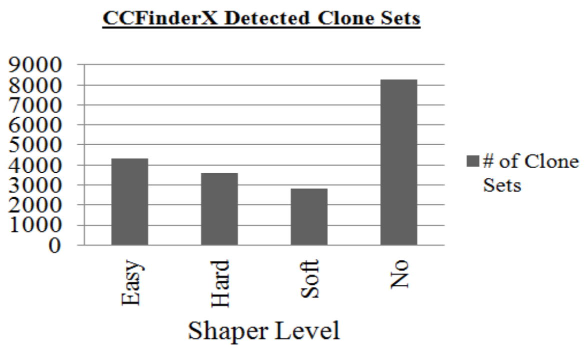


Fig 6: No. of found clones using CCFinderX by varying Shaper

A file is selected randomly from VLC media player source files and variations in results are noted. The access.c is that chosen file. This file has 557 LOC and 349 SLOC.

Experiment was conducted using SolidSDD clone detector with following default values of parameters.

CS= 35; LG = 2; CG = 2.5; GD = 0.1

Output was as below:

CCFinderX also experimented with the same file with following parameters with default values.

- CL = 50; MTK = 12; PM = Yes; SH = soft.

We have chosen three parameters from file view for comparison purpose because these three parameters are note-worthy and fully exist in both tools. The first and foremost parameters is Count of detected clones, second is Clone coverage and last one is Number of files that are involved in cloning relation

As definitions of all these terms are cited in section 3 it can be easily seen that CLN in CCFinderX and # of Clone in SolidSDD are count of detected clones.% clone in SolidSDD and CVR in CCFinderX are similar i.e. percentage of tokens/statements that are covered by clone code. Similarly Fan-Out in SolidSDD and NBR in CCFinderX both mean number of files that have a particular clone code fragment. A comparison of these parameters is shown in table 6.

This comparison is presented graphically below.

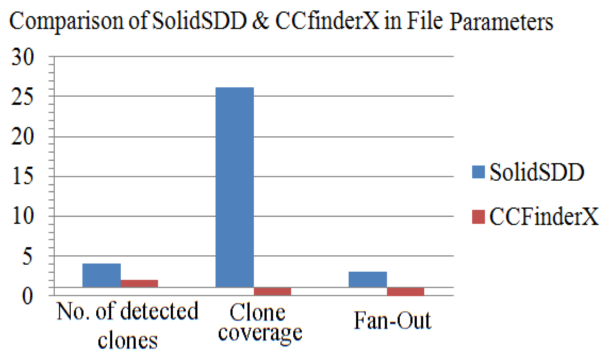


Fig 7: CCFinderX and Solid SDD File perspective comparison.

After analyzing the code in File perspective the code is also analyzed in Clone perspective. Experiment using SolidSDD is conducted with same file and parameters as in File perspective.

- CS= 35; LG = 2; CG = 2.5; GD = 0.1

Output is shown in table7.

CCFinderX is also tested in Clone perspective with the same file namely access.c and parameters as in File perspective i.e.

- CL = 50; MTK = 12; PM = Yes; SH = soft

Output is given in table8. In clone perception, Number of instances of particular clone, counts of files that include a clone code are selected for comparison purpose.

POP in CCFinderX and # of instances (No. of instances) in SolidSDD both mean count of that particular clone. So, both of these are chosen for mutual comparison. NIF in CCFinderX and Fan out in SolidSDD are alike according to their referenced definitions hence they are compared jointly. Results of compared parameters are in table 9.

4.3 Discussion on Results

SolidSDD and CCFinderX both tools offer easy to use interface for detecting clones. They equally provide graphical output in form of visual charts and scatter plot for viewing clones scatterness in the source code. There are minor differences in both tools outcomes.

aSolidSDD and CCFinderX found equal number of clones when consider the whole software. But variations lie when analysis narrowed down to individual files of software. As can be seen from the above comparative tables CCFinderX detects 2 clone instances while SolidSDD spots 4 occurrences in access.c. Both tools have only one common instance of clone between them. According to SolidSDD that particular instance also exists in another file i.e. tcp.c which is also true according to CCFinderX findings. SolidSDD skips 1 clone that is discovered by CCFinderX while CCFinderX do not consider the 2 clones which are included in SolidSDD detection results. Furthermore, CCFinderX takes less time for detection as compared to SolidSDD. When evaluating manually it is found that SolidSDD requires almost double time as compare to CCFinderX for computing clones.

SolidSDD has more convenient configuration as compared to CCFinderX while CCFinderX is more mature tool and gives exceptionally detail information about clones in all perspectives. So, CCFinderX is better than SolidSDD in terms of providing information for clone analysis and taking shorter time frame.

Table 1. Characteristics of Solid SDD and CCFinderX.

Attribute Name	SolidSDD	CCFinderX
Platform	Microsoft Windows(XP, Vista, 7)	Ubuntu i386, Windows Vista 32-bit/XP and later.
Supported Languages	C, C++, C#, Java and H	COBOL, cpp, java C#, plaintext, visual basic
Approach	Textual	Token Based
External Dependency	None	For windows Python2.6
Memory	1GB minimum, 4 GB advised;	None
Availability	Free evaluation licensed for 1 month	Freeware
Output Method	Reports and visual charts	Scatter plot
UI	GUI	Batch Tool (CLI) with GUI(GemX)
IDE Support	NO	NO
Metrics	File and Clone Metric	Clone, File and Line based metric

Table 2. Count of clones in VLC media player using SolidSDD

Attribute	Value
Found Duplicates	48
Execution Time	35

Table 3. Count of clones in VLC media player using CCFinderX

Attribute	Max Value
LEN	22462
CLN	48
NBR	26
RSA	0.992
RSI	0.999
CVR	1.000
RNR	1.000

Table 4. Count of clones in VLC media player using CCFinderX

Attribute	Value
# of Clones	4
Fan Out	3
% Cloned	26.23
ID renaming	Yes

Table 5. CCFinderX File Perspective with access.c

Attribute	Value
File-ID	208
LEN	1870
CLN	2
NBR	0
RSA	0
RSI	0.117
CVR	0.117
RNR	0.843

Table 6. CCFinderX File Perspective with access.c

Attribute	SolidSDD	CCFinderX
No. of detected clones	4	2
Clone coverage	26.23	0.117
Fan-Out	3	0

Table 7. SolidSDD Clone Perspective with access.c

Attribute	Clone ID		
	45	**17**	**44**
#instances	2	2	3
Fan out	2	2	3
Length	56	36	35
Total gap	5.0	5.0	7.0
ID renaming	Yes	Yes	Yes
#Ren	8	6	12
#Ren ID	8	6	12
Norm(#Ren)	0.71	0.83	0.57
Norm(#IDs)	0.71	0.83	0.57

Table 8 CCFinderX Clone Perspective with access.c

4.4 Attribute	4.5 Clone Set ID	
	4.6 1495	**4.7 1651**
4.8 LEN	4.9 70	4.10 53
4.11 POP	4.12 2	4.13 2
4.14 NIF	4.15 1	4.16 1
4.17 RAD	4.18 0	4.19 0
4.20 RNR	4.21 0.686	4.22 0.472
4.23 TKS	4.24 15	4.25 14
4.26 LOOP	4.27 0	4.28 0
4.29 COND	4.30 1	4.31 0
4.32 McCabe	4.33 1	4.34 0

Table 9. : SolidSDD and CCFinderX Clone Perspective comparison with access.c

4.35 Attribute	4.36 SolidSDD	4.37 CCFinderX
4.38 No. of instances of that clone	4.39 2	4.40 2
4.41 Fan-out	4.42 2	4.43 1

5. CONCLUSIONS & FUTURE WORKS

This paper demonstrates that clone code detection tools show many variations in results. It is revealed here that some tools may show similar results at whole but after narrowing down the analysis variations can be found. Furthermore, this work also identifies that careful selection of input parameters are necessary when comparing two different approaches.

There is need to define a consistent methodology for analyzing clones. A benchmark can be developed for evaluating and comparing clone code detection tools. Cloning is considered one of the bad codes smells which ultimately degenerate the software quality. As refactoring is regarded as one of best cure for this smell so in parallel, we are trying to do a qualitative based research in this perspective. Objective of this research is to check and understand those practices which are in process in industry. We have developed a questionnaire in which we are going to study relationship between quality attributes and its impact on refactoring. Also a prototype is in the development stage.

6. REFERENCES

[1] Fowler M, Beck K, Brant J, Opdyke W, Roberts D; *"Refactoring improving the design of existing code*"; Addison-Wesley, Boston; USA, (1999)

[2] Juergens E.; Deissenboeck F.; Hummel B, Wagner S; "Do code clones matter?" *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on* , vol., no., pp.485,495, 16-24 May 2009

[3] Kim M, Sazawal V, S.; Notkin D, Murphy G; "An empirical study of code clone genealogies"; *ESEC/FSE-13Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering.* pp 187-196

[4] Kaur P, Kaur H, Kaur R; "Comparison of Clone Detection Tools: CONQAT and SolidSDD". *International Journal of Advanced Research in Computer Science and Software Engineering* Volume 2, Issue 5 May 2012.

[5] Rysselberghe F A, Demeyer S; "Evaluating clone detection techniques"; Proceedings ELISA'03 *(International Workshop on Evolution of Large-scale Industrial Software Applications),* pages 25–36; Vrije Universiteit Brussel, September 2003.